An Overview of Parallelism Exploitation and Cross-layer Optimization for Big Data Transfers

Eun-Sung Jung, *Member, IEEE*, and Rajkumar Kettimuthu, *Senior Member, IEEE*

Abstract—**The data produced by sensors are expected to grow exponentially in the next several years. In particular, e-science applications need to move big data for remote analysis or data distribution, which is required to efficiently utilize distributed resources such as supercomputers, data centers, scientific instruments and the network that connects these facilities. In this article, we evaluate prior work on data transfer over wide-area networks focusing on parallel and cross-layer optimization methods. The cross-layer optimization methods reduce the overhead incurred by many layers or improve the performance of data transfer by appropriately using the information of layers in a holistic way. The goal of this study is to help researchers focus more on unexplored areas, e.g., multipath-aware network protocols, to better handle big data.**

I. INTRODUCTION

In the era of big data, both compute-intensive computing, which is usually called extreme-scale computing, and data-intensive computing are crucial in high-performance computing communities. Extreme-scale computing seeks for novel solutions scalable up to millions of cores, which will be realized in near future. Likewise, data-intensive computing is becoming more important as the data growth rates in most areas of sciences as well as general information technology areas are projected to dramatically increase up to petabytes per application. Consequently, data movement in data-intensive computing is one of the key factors influencing overall performance of such data-intensive workflows.

Data movements happen in micro and macro level among compute, network, and data storage elements. In micro-level data movements, circuit level studies such as network on chip are exemplary. In macro-level data movements, data movements among independent compute/storage boxes are typical research issues. In this article, we focus on data movement mechanisms and techniques from the perspectives of distributed workflow execution. The distributed workflow are becoming prevailing in scientific computing where experimental facilities such as DOE light source facilities, observational instruments such as telescopes, and computing facilities, e.g., supercomputers, are usually geographically distributed, which makes data movements over wide-area networks (WAN) inevitable.

There has been a huge amount of work to improve data movement rates with the emergence of new network technologies, new storage technologies, and so on. In particular, the exploitation of parallelism inherent in systems involved in the end-to-end data movement path has been successful in scaling data movement throughput. One simple example is striping data among multiple physical disks as in redundant array of independent disks (RAID). We first present what system components are involved in the end-to-end data movement in distributed data-intensive workflow based on a layered model. We then give an overview of prior work in the context of parallel computing so that readers can see which layers are fully utilizing parallelism and which layers still have room for further improvement.

In addition to advances through parallel computing in each layer, cross-layer optimization has been another approach taken to either reduce unnecessary overhead through bypassing layers or apply other layers' information to efficient decision making. One simple example is combining disk throughput information and network bandwidth information to determine the number of TCP streams required to maximize the data movement throughput.

The remainder of the article is organized as follows. In Section II, we describe preliminaries on data movements in the context of distributed data-intensive workflow. In Section III, we present a layered model, and categorize and elaborate prior work regarding parallel computing for big data transfer based on the model. In Section IV, we give an overview on cross-layer optimization approaches for efficient data movement. In Section V, we briefly summarize the article.

II. OVERHAUL OF DATA TRANSFER

The end-to-end data transfer spans from disks via network to disks, and goes through many other hardware or software components related to the data transfer. Figure 1 captures the major system components participating in data transfers often happening in distributed workflow. A data transfer is initiated by a user or a workflow management module. The data transfer request is eventually carried out by hosts. In case of large computing facilities such as Argonne Leadership Computing Facility (ALCF), dedicated computers, called data transfer nodes (DTNs), are used to switch between network and disks.

The host sending data reads data from disks and sends them to another host over networks. The host receiving data does the same series of work in reverse order. For a storage operation such as a read or write, the request is processed by several layers of system software such as file system, logical device drivers, and hardware device drivers. Current storage systems are massively parallel rather than a single disk and may be front-ended by a complex parallel file system (PFS) stack in case of large high performance computing facilities. Typically, the PFS is shared by compute nodes and data transfer nodes. A single application I/O request may be split into several "physical" I/O requests, and these requests must pass through the interconnection switch that connects the host bus adapter on the computer system and the disk array controller. The disk array controller takes the I/O request and creates one or more I/O requests to the individual disks in the array. The data is then transferred between the disk drive and the application memory space with multiple buffering along the way.

The sender may want to apply data preprocessing filters to data streams before it send them out to networks. The typical data preprocessing filters are compression and encryption, which are independent of internal data processing of applications. The multiple cores in the host, which are indicated by capital C in the Figure 1, are used to execute data preprocessing filters in parallel. Multiple network interface cards (NICs) can be engaged to achieve higher throughput beyond the capacity of each NIC. The network paths that transfer data between two sites are contingent on network properties. For example, if network connecting the sites is a dynamic circuit network supporting deterministic network paths with bandwidth guarantee, static network paths with fixed bandwidth can be set up for data transfers between two hosts. The workflow management module may engage multiple hosts for data transfers between two sites if necessary. We describe detailed previous research work in each layer in the following section.

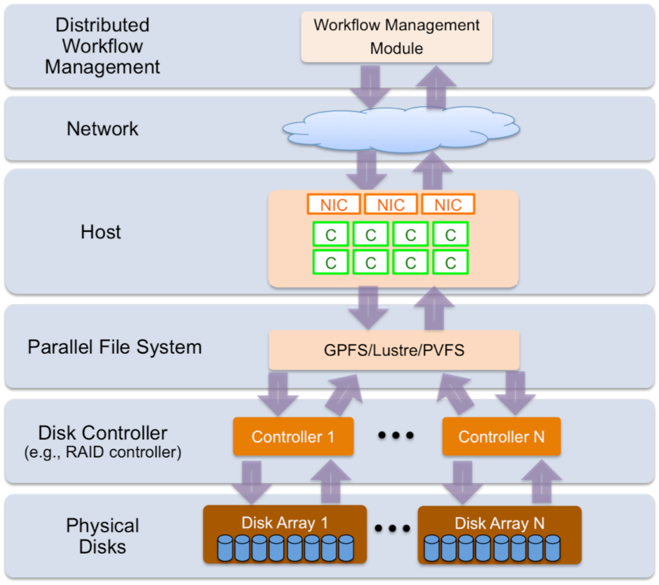

Fig. 1. System overhaul regarding data transfer

III. Parallelism Exploitation

In general, parallel computing can be classified into three categories: data parallelism, task parallelism, and pipeline parallelism. We use this criterion to classify previous studies and map those representative studies in Table 1. The layers in Table 1 are extracted from system layers in Figure 1. Four layers are defined; distributed workflow, network, application, and storage layers. The distributed workflow layer deals with data transfer among applications over networks, which are typically LAN or WAN. The network layer deals with network routing and protocols for data transfers. The network layer is further divided into two sub-layers, middleware and network protocol. The application layer deals with orchestrating data streams in the application level. The application layer is also further divided into two sub-layers, data stream transporting and data stream processing layers. The storage layer deals with disk request handling from applications down to physical disks. The storage layer also has two sub-layers, file system and storage system layers.

A. Data Parallelism

Data parallelism refers to multiples instances of a single task running on different datasets. Accordingly, distributing datasets to multiple computing nodes is a main issue in data parallelism.

In the storage layer, parallel file system (PFS) and high-performance storage system (HPSS) are prominent in terms of parallelism. IBM's General Parallel File System (GPFS) [9], Luster, and PVFS are most adopted PFSs. Such parallel file systems have multiple file servers and metadata servers, both of which provide file system I/O services to file system clients. Exceptionally, PVFS has a single metadata server called MGR, which is featured by one-time MGR access by file system clients to avoid MGR bottleneck. The number of file system clients can amount up to millions, and the file servers and MDSs are running in parallel to achieve

TABLE I
Parallelism in Big Data Transfers

Layer\Parallelism		**Data Parallelism**	**Task Parallelism**	**Pipeline Parallelism**
Distributed Workflow		Data flow based parallel workflow scheduling [1,2]	Control flow based workflow scheduling [1]	Overlapping execution with data staging [2]
Network	*Middleware*	Multipath (OSCARS) [3]		
	Network Protocol	MPTCP [4]		GridFTP-pipeline [5]
Application	*Data Stream Transporting*	GridFTP-parallel data streams [5]	GridFTP-cluster-to-cluster(striped) server [6]	
	Data Stream Processing	Parallel Compression [7]	Parallel Data Stream Processing [1,8]	
Storage	*File System*	PFS-parallel data access [9]	PFS-parallel file service [9]	IOFL-pipeline data transfer [10]
	Storage System	HPSS-striped data storage [11]	HPSS-concurrent servers [11]	

extreme scalability and throughput in HPC systems. HPSS [11] is storage software developed by IBM in conjunction with DOE national labs over a decade. In particular, the GPFS architecture is built on shared disks and provides parallel data access for multiple file system nodes through data parallelism. Similarly HPSS achieves high aggregate disk throughput by appropriately using data parallelism such as striped data storage.

In the application layer, GridFTP uses parallel data streams to transport portions of a large file in parallel [5]. The data processing ahead of transporting such as compression [7] and encryption can also be performed in parallel. Bicer *et al.* [7] proposes an online parallel compression/decompression framework for data-intensive applications, which is shown to benefit data-intensive applications by amortizing I/O time. Data compression utilizing CPU resources helps reduce network bandwidth consumption and improve the overall data transfer throughput especially in case of network bottleneck.

In the network layer, many studies including congestion control and stability [4] have been done for multi-path TCP (MPTCP) to achieve higher data transfer throughput beyond a single path TCP. MPTCP is now concrete proposal for IETF. However, MPTCP by itself cannot know or guarantee the use of multiple independent network paths between the source and the destination. MPTCP just tries to use multiple network interfaces in a multi-homed host while establishing the TCP connections. Dynamic circuit network services such as OSCARS [3] support multiple network paths in network infrastructure level. In OSCARS, a centralized manager sends controls messages switches/routers to set up deterministic network paths over wide-area networks. The virtual circuits in OSCARS are managed in layer 3 network level using multi-protocol label switching (MPLS) and resource reservation protocol (RSVP). The separation of control plane and data plane in OSCARS is similar to the design principle of the emerging software-defined networking (SDN) technology.

In the distributed workflow layer, massively parallel workflow scheduling for a large number of datasets corresponds to data parallelism. Several workflow management systems supporting data flow control including Swift [2], Wings for Pegasus [1], and Askalon [1] can distribute datasets to multiple distributed computation sites for the same data processing task.

B. *Task Parallelism*

Task parallelism refers to multiple tasks concurrently executing on different cores. Accordingly, distributing tasks to multiple computing nodes is a main issue in data parallelism.

The distributed lock manager in PFS such as GPFS [9] and Lustre overcomes the bottleneck of a centralized lock manager so that it can scales up to a large number of nodes, which exploits task parallelism. Likewise, the HPSS architecture has multiple concurrent servers with diverse functions such as metadata management and migration, which utilizes task parallelism.

In the application layer, GridFTP cluster-to-cluster architecture [6] provides striped or interleaved data transfer across multiple nodes in clusters. Although the architecture is more close to data parallelism from a perspective of one file transfer, it can be regarded as an architecture using task parallelism in a sense that multiple data transfer requests can be handled in different sets of nodes in clusters. Regarding data stream processing, general parallel data streaming platforms such as IBM InfoSphere [8] inherently exploit task parallelism by mapping data stream task graphs onto multicores in a server farm. Even though current implementation such as Globus XIO [14] assumes a serial data processing task session, e.g., encryption followed by compression, more complex data processing tasks, e.g., parallel execution of compression and checksum computation followed by a merge of each result, can benefit from task parallelism as in IBM InfoSphere.

In the distributed workflow layer, usual workflow schedulers [1] utilize task parallelism inherent in task graphs, which formally specify workflows, by running tasks without precedent constraints concurrently on different compute resources.

C. *Pipeline Parallelism*

Pipeline parallelism refers to a succession of tasks, each of which runs independently, that a stream of data passes through. Accordingly, distributing pipeline tasks to multiple computing nodes and determining appropriate data exchange size among pipeline tasks are main issues in pipeline parallelism.

Ohta *et al.* [10] propose pipeline transfer optimization for the I/O forwarding layer, which is used to eliminate I/O bottleneck from computing nodes to storage systems by reducing the number of I/O requests. The basic idea of pipeline transfer optimization is overlapping application I/O requests and file system I/O requests in which I/O requests pass through client I/O, IOFL, and PFS layers.

In the network layer, GridFTP enhances standard FTP through data channel reuse and pipelining control message exchange and file transfer. The standard FTP can send only one file once data channel is established. GridFTP makes data channel reusable for multiple file transfers. Regarding file transfers, control message exchange precedes a file transfer, and standard FTP cannot overlap control message exchange with a file transfer. In case of multiple small file transfers, this mechanism exacerbates overall file transfer time due to increased unnecessary overhead, which is called the lots of small files (LOSF) problem. By overlapping control message exchange with actual file data transfer, GridFTP can expedite multiple small file transfers [5].

In the distributed workflow layer, Jung *et al.* [2] propose algorithms and mechanisms to overlapping data transfers and computation. In the context of distributed workflow execution, multiple jobs may be concurrently executed across multiple computation sites. Without loss of generality, a job can be simplified as a sequence of three phases; stage-in, execution, and stage-out. In the stage-in phase, the input data for the computation is moved to the computation site and computation on the data follows. The output data of the

TABLE II
CROSS-LAYER OPTIMIZATION

Layer		Cross-layer Approach					
Distributed Workflow			Multipath /Network-centric workflow scheduling [12]				
Network	*Middleware*			MPTCP based on determined paths			End-system aware data movement [15]
	Network Protocol					RDMA [13], Globus XIO [14]	
Application	*Data Stream Transporting*						
	Data Stream Processing				PFS & Data Processing embedded in Storage Systems		
Storage	*File System*	PFS embedded in Storage Systems [9]					
	Storage System						

computation is sent out to the next distributed sites on which descendent jobs are scheduled. By overlapping those three phases, the execution time of distributed parallel jobs is significantly improved [2].

D. Discussion

In most areas, parallel computing techniques have been successfully applied, and it led to significant performance improvements. However, current data transfers are being carried out either by compute nodes, which means data transfer is not first priority jobs in those nodes, or by DTNs, which are high-end computers dedicated for data movement. This may have been enough for current data transfers, but high-performance data transfers with increased amount of data and stricter quality of service requirements will require innovative data transfer mechanisms and infrastructures. For example, current Globus XIO framework on which GridFTP is built, does not fully exploit parallelism as much as IBM InfoSphere does. This may be because current needs for data preprocessing are limited to several basic functions such as compression and accordingly thin software layer is preferred. With increased demand for data preprocessing at the level of data transfers and the emergence of cheaper multicore systems, DTNs may need more sophisticated parallel computing mechanisms.

IV. CROSS-LAYER OPTIMIZATION

The purpose of cross-layer optimization is twofold: 1) Overhead reduction and 2) Impedance matching. Some layers in Figure 1 can be bypassed to reduce overhead introduced by deep layers. For overhead reduction in data transfer, thorough analysis has to be done to understand the tradeoff between the flexibility that a layer brings in versus the overhead. Cautious decision can be made to eliminate overhead introduced by multiple layers by merging/bypassing one or more layers or making a layer as thin as possible. The second purpose is impedance matching, which tries to avoid overprovisioning or underprovisioning of resources while achieving the maximum data transfer throughput. For example, if the maximum disk throughput obtained from the storage system is limited by 100 MB, we do not have to request for a network path with 200 MB bandwidth.

A. Reducing Overhead

In the storage layer, one approach to reduce overhead is by merging two sub-layers, i.e., file system and storage system, is GPFS Native RAID [9]. GPFS Native RAID embeds a GPFS I/O node onto the external RAID controller such that the single merged layer brings declustered RAID rebuild management function into GPFS, which is much faster than the previous two-layer approach, and eliminates the need for a storage controller. Even though this study is not directly related to data transfers, this study is meaningful in a sense that data transfer during disk failure can maintain throughput comparable to data transfer during normal operation in RAID disks.

Across the storage, the application, the network layers, the remote direct memory access (RDMA) [13] has been proposed to reduce overhead incurred from redundant copies at each protocol layer and host CPU involvement by direct memory-to-memory data movement from one host to another. RDMA does not require CPU or caches by enabling network adaptors to transfer data from/to application buffers, which bypasses file system and network protocol. Kissel *et al.* [13] shows that RDMA over WAN performs significantly better than TCP with a fraction of CPU power.

B. Impedance Matching

The Globus eXtensible I/O (XIO) [14] system is a unified framework integrating disk I/O, data processing, and network I/O. The Globus XIO architecture is a software layer model where users can put as many layers as they need for their purpose. For example, if a user needs to send compressed data using UDT protocol, the user can just configure the Globus XIO stack with a compression layer and a UDT layer on top of it. The Globus XIO itself does not address any cross-layer optimization problem, but provides a solid abstract model, which any optimization techniques can be developed based on. Jung *et al.* [14] addresses the cross-layer optimization problem by allocating variable numbers of threads to different layers in order to prevent a certain layer from being a bottleneck while improving performance. For example, say that two layers, a compression layer and a TCP layer, are involved. The compression layer is compute-intensive compared with TCP layer and more number of threads needs to be allocated for the compression layer.

Spanning from the network layer to the distributed workflow layer, multipath/network-centric workflow scheduling [12] is one example of cross-layer optimization approaches. Basically, multipath/network-centric workflow scheduling takes into account compute resources and network resources at the same time. Whereas general workflow scheduling first schedule compute resources and later determine network paths, multipath/network-centric workflow scheduling tries to determine compute resources and network paths at the same time, even further exploring multiple network paths for data transfer between two sites. In this way, the number and bandwidth of network paths are optimized for overall workflow execution time (makespan), which means data transfer throughput is well matched with computation site throughput.

In a broader sense, there have been few studies to optimize throughput/impedance of all system components involved in a complete end-to-end data transfer path. Jung *et al.* [15] models all system components using a graph data structure and models throughput of a system component as a bandwidth capacity function of a link in the graph model. Based on the graph model, linear programming based optimization techniques have been develop to maximize overall throughput.

C. Discussion

As system architecture becomes more deep and heterogeneous, the cross-layer optimization problems get more complex and even intractable. The overall system architecture should be carefully designed according to the design goal. For example, latency or jitters are crucial issues for real-time visualization applications. In such cases, more layers for flexibility are not good options, and reduced layers and cross-layer optimization techniques should be instead sought after. This article is to give an overview of current cross-layer optimization techniques and may help identify unexplored research areas. The dark shade squares in Table II show examples of some ideas. First, MPTCP may be more tuned for performance enhancement if we know how many paths are available and the characteristics such as delays are known ahead. The other example can be data processing functions as well as PFS are also embedded in storage systems. We know similar ideas are being proposed such as ‘Smart SSD’, which pursue the key concept of putting computation near data to reduce data movement bottleneck, One example of Smart SSD proposals is query processing on Smart SSDs. However, those ideas should be viewed from the perspectives of overall data transfer architecture, too.

V. Summary

In this article, we give an overview of big data transfer from the perspectives of parallelism exploitation and cross-layer optimization. We present a system layer model to classify the prior work in the literature and identify the unexplored research area. Exploiting parallelism has improved big data transfer performance drastically and some areas need more tuning and integration work to apply the state of the art to them. The cross-layer optimization relatively has not been extensively researched yet, and remains as an active research issue.

Acknowledgment

This work was supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research, under Contract DE-AC02-06CH11357.

References

[1] E. Deelman, D. Gannon, M. Shields, and I. Taylor, “Workflows and e-Science: An overview of workflow system features and capabilities,” *Future Generation Computer Systems*, vol. 25, no. 5, pp. 528–540, May 2009.

[2] Eun-Sung Jung, Ketan Maheshwari, and Rajkumar Kettimuthu, “Pipelining/Overlapping Data Transfer for Distributed Data-Intensive Job Execution,” in *10th International Workshop on Scheduling and Resource Management for Parallel and Distributed Systems (SRMPDS)*, 2013.

[3] C. Guok, D. Robertson, M. Thompson, J. Lee, B. Tierney, and W. Johnston, “Intra and Interdomain Circuit Provisioning Using the OSCARS Reservation System,” in *3rd International Conference on Broadband Communications, Networks and Systems, 2006. BROADNETS 2006*, 2006, pp. 1–8.

[4] H. Han, S. Shakkottai, C. V. Hollot, R. Srikant, and D. Towsley, “Multi-path TCP: A Joint Congestion Control and Routing Scheme to Exploit Path Diversity in the Internet,” *IEEE/ACM Trans. Netw.*, vol. 14, no. 6, pp. 1260–1271, Dec. 2006.

[5] E. Yildirim, J. Kim, and T. Kosar, “How GridFTP Pipelining, Parallelism and Concurrency Work: A Guide for Optimizing Large Dataset Transfers,” in *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, 2012, pp. 506–515.

[6] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster, “The Globus Striped GridFTP Framework and Server,” in *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, Washington, DC, USA, 2005, p. 54–.

[7] T. Bicer, J. Yin, D. Chiu, G. Agrawal, and K. Schuchardt, “Integrating Online Compression to Accelerate Large-Scale Data Analytics Applications,” in *2013 IEEE 27th International Symposium on Parallel Distributed Processing (IPDPS)*, 2013, pp. 1205–1216.

[8] A. Biem, E. Bouillet, H. Feng, A. Ranganathan, A. Riabov, O. Verscheure, H. Koutsopoulos, and C. Moran, “IBM Infosphere Streams for Scalable, Real-time, Intelligent Transportation Services,” in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, New York, NY, USA, 2010, pp. 1093–1104.

[9] F. B. Schmuck and R. L. Haskin, “GPFS: A Shared-Disk File System for Large Computing Clusters,” in *Proceedings of the Conference on File and Storage Technologies*, Berkeley, CA, USA, 2002, pp. 231–244.

[10] K. Ohta, D. Kimpe, J. Cope, K. Iskra, R. Ross, and Y. Ishikawa, “Optimization Techniques at the I/O Forwarding Layer,” in *2010 IEEE International Conference on Cluster Computing (CLUSTER)*, 2010, pp. 312–321.

[11] HPSS - High Performance Storage Systems, [Online]. Available: http://www.hpss-collaboration.org/index.shtml.

[12] E.-S. Jung, S. Ranka, and S. Sahni, “Workflow scheduling in e-Science networks,” in *2011 IEEE Symposium on Computers and Communications (ISCC)*, 2011, pp. 432–437.

[13] E. Kissel and M. Swany, “Evaluating High Performance Data Transfer with RDMA-based Protocols in Wide-Area Networks,” in Proceedings of the 2012 IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems, Washington, DC, USA, 2012, pp. 802–811.

[14] Eun-Sung Jung, Rajkumar Kettimuthu, and Venkatram Vishwanath, "Cluster-wise disk-to-disk transfer with data compression over wide-area networks", ANL techreport.

[15] Eun-Sung Jung, Rajkumar Kettimuthu, and Venkatram Vishwanath, "Toward optimizing disk-to-disk transfer on 100G networks," in *IEEE International Conference on Advanced Networks and Telecommunications Systems*, 2013.

BIOGRAPHIES

Eun-Sung Jung (esjung@mcs.anl.gov) is a postdoctoral researcher in the Mathematics and Computer Science Division at Argonne National Laboratory. He earned a Ph. D. in the Department of Computer and Information Science and Engineering at the University of Florida. He also received B.S. and M.S. degrees in electrical engineering from Seoul National University, Korea, in 1996 and 1998, respectively. He also held a position of a research staff member at Samsung Advanced Institute of Technology from 2011 to 2012. His current research interests include cloud computing, network resource/flow optimization, and real-time embedded systems.

Raj Kettimuthu (kettimut@mcs.anl.gov) is a project leader in the Mathematics and Computer Science Division at Argonne National Laboratory and a fellow at University of Chicago's Computation Institute. His research interests include transport protocols for high-speed networks; research data management in distributed systems; and the application of distributed computing to problems in science and engineering. He is the technology coordinator for Globus GridFTP, a widely used data movement tool. He has published over 60 articles in parallel, distributed and high performance computing. He is a senior member of IEEE and ACM.